Authentication is the act of confirming that users are who they claim to be. Authenticating a claim might be done using ASP.NET security, Active Directory, Windows Identity Foundation, or a custom implementation.

DevForce has built-in hooks to support the authentication scheme of your choice, and includes out-of-the-box support for ASP.NET security.

Unless your application allows "guest" access, when your application starts you should obtain user credentials. These might be in the form of a userid and password (when using ASP.NET security this is called Forms authentication), or it might mean obtaining an "ambient" credential, such as the current Windows userid or *HttpContext.Current.User*.

In DevForce, an EntityManager must be "logged in" to the EntityServer in order to access data or services. As a developer you have control over how and when this login occurs. You determine the credentials your application accepts, you call a *Login* method to authenticate the user, and you write the server code to perform the authentication via the *IEntityLoginManager*.

When you determine a user is authenticated, you provide that information to DevForce in the form of an IPrincipal. If you're not familiar with the *IPrincipal* interface, it, along with the *IIdentity* interface, is a flexible means of holding information on who a user is and their roles. It does not hold user credentials or claims, but does indicate how the user was authenticated. The DevForce implementation of these interfaces is the *UserBase*.

On the server DevForce will tuck this *IPrincipal* away, and return a security token to the client. This token includes an encrypted credential, encrypted using the EntityServer's specific "session encryption key". The security token is transmitted with every request to the server, where DevForce first validates the token before proceeding with the request. In Azure and other load balanced environments, the encrypted credential in the token allows the user to be authenticated by any server having the same session encryption key. (This key is configurable and we'll see more on this key later.) Once logged in, the server keeps track of each user's last access, and will time the session out after a configurable period of inactivity.

Guest Users

A guest user is an anonymous user -- one who has not provided a credential and is therefore not authenticated. We noted above that an EntityManager must be logged in to an EntityServer, and that every request to the server is accompanied by a token which is validated before proceeding. How does this work with anonymous users? If you do not call a Login method and supply a credential, DevForce will do it for you when necessary. DevForce first determines whether you have disabled anonymous access via a configuration setting; if not, a *UserBase* is constructed for the guest user indicating the user is not authenticated and has no assigned roles.

The DevForce default is to allow anonymous users, but it is a very good idea to disable this feature unless your application intentionally allows it. When anonymous logins are disabled, an exception is thrown for the Login attempt, and no further access to the EntityServer is permitted from the EntityManager.

To disable guest access, set the *AllowAnonymousLogin* flag to false in the web.config or server .config file:

```
<objectServer>
  <serverSettings allowAnonymousLogin="false" />
</objectServer>
```